2023. 08. 31
금오공과대학교 SSL 세미나

Kotlin Spring

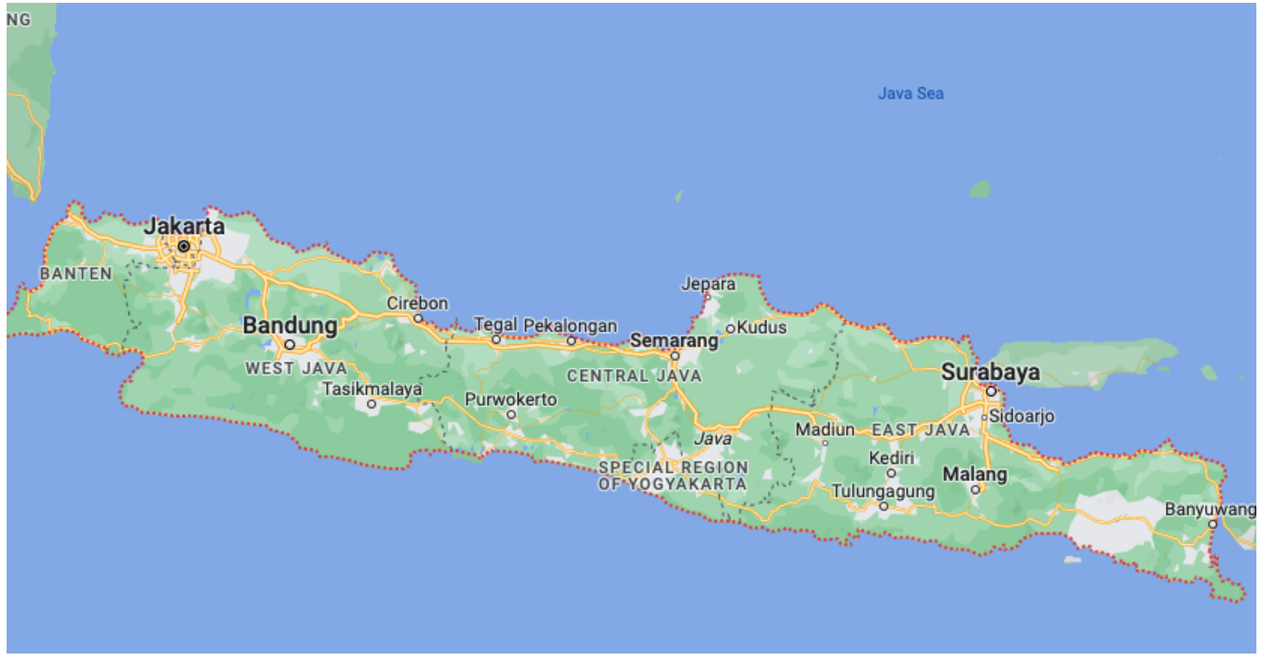
박준수
[취준생/만 24세]

Contents.

01

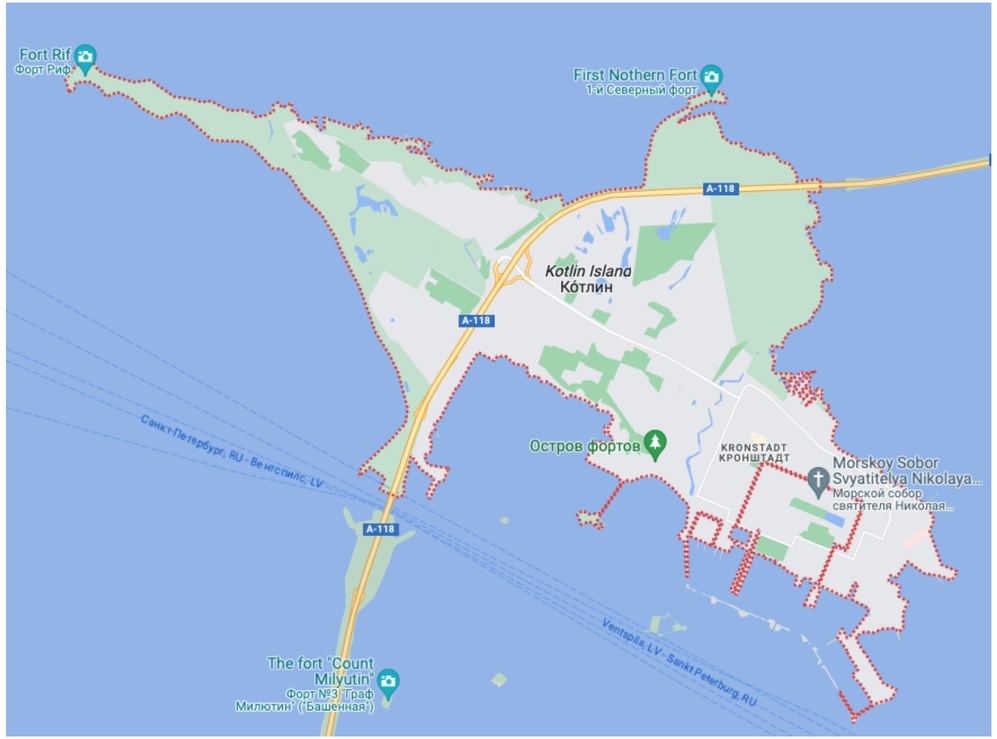
Kotlin

- 코틀린이란?
- 기본 문법



인도네시아 자바 섬

러시아 코틀린 섬



Kotlin

JET BRAINS
Developed by JetBrains & Open-source Contributors

Apache License 2.0

OOP, FP 기법을 모두 제공하는 멀티패러다임 언어

문법이 간결하고 실용적이며 안전함

기존의 언어들과의 상호 운용성을 중시

Kotlin

Kotlin/JVM
Kotlin/JS

Common JVM JS Native

JVM, Android 외 NodeJS, JS, Native
다양한 플랫폼에서 사용가능

android

2019 Google's preferred language for
Android app development

Variables

```kotlin
fun main(args: Array<String>) {

    // inferred
    var name = "jun"

    // explicit
    val hisName : String = "jay"

    // null safe
    name = null
    val hmmCanBeNull: String? = null

    name = "jun-su"
    println("$name; $hisName; $hmmCanBeNull")

}
```

mutable -> **var**

read-only -> **val**

String Template

: $variable

${variable + 2}

Data classes

Kotlin has data classes which are particularly useful for storing data.
They come automatically with additional member functions.

These member functions allow you
to easily print the instance to readable output,
compare instances of a class,
copy instances, and more.

As these functions are automatically available, you don't have to spend time
writing the same boilerplate code for each of your classes.

출처: https://kotlinlang.org/docs/kotlin-tour-classes.html#data-classes

Data classes

이것을 DTO로 사용하면 좋을거 같은 느낌

Kotlin has data classes which are particularly useful for storing data.
They come automatically with additional member functions.

These member functions allow you
to easily print the instance to readable output, .toString()
compare instances of a class, .equals()
copy instances, and more. .copy()

As these functions are automatically available, you don't have to spend time
writing the same boilerplate code for each of your classes.

반복적이지만 일상적인 코드

자동으로 만들어지니까 얼마나 시간이 절약될까!

출처: https://kotlinlang.org/docs/kotlin-tour-classes.html#data-classes

Data classes

```kotlin
data class User(
    var name: String,
    val id: Int
)

class General(
    val name: String,
    val id: Int
)
```

```kotlin
val a = General( name: "Jason",  id: 990820)
println(a == General( name: "Jason",  id: 990820))
println(a)

println()

val b = User( name: "Jay", id: 19990820)
println(b == User( name: "Jay", id: 19990820))
println(b)

println(b.copy())
println(b.copy(id = 1999))
```

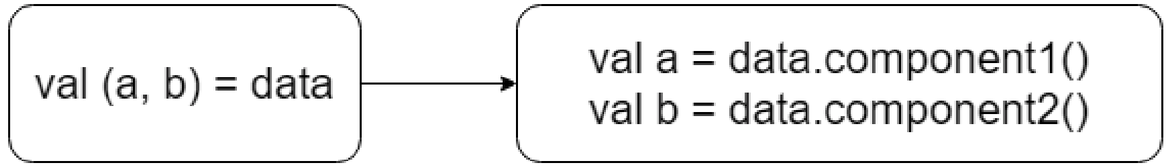
```
false
General@1d251891

true
User(name=Jay, id=19990820)
User(name=Jay, id=19990820)
User(name=Jay, id=1999)
```

Data classes

```kotlin
val userList = listOf(
    User( name: "준수",  id: 19990820),
    User( name: "Jason",  id: 19990821),
    User( name: "Jay",  id: 19990822),
)

for ((name, id) in userList){
    println("이름: $name, ID: $id")
}
```

```
이름: 준수, ID: 19990820
이름: Jason, ID: 19990821
이름: Jay, ID: 19990822
```

구조 분해 선언



componentX()

Null Safety 모든 변수를 not-null로 정의

```
fun checkString(maybeString: String?): String = maybeString ?: "I THINK IT'S NULL"
fun lengthString(maybeString: String?): Int? = maybeString?.length
fun lengthStringDef(maybeString: String?): Int = maybeString!!.length
```

```
println("=== A ===")
var stmt: String? = "SELECT city AS 도시, COUNT(city) AS 집계\n" +
        "  FROM user\n" +
        "    WHERE user.age >= 18\n" +
        "      GROUP BY city\n" +
        "        HAVING city >= 'b'\n" +
        "          ORDER BY city"
println(checkString(stmt))
println(lengthString(stmt))
println(lengthStringDef(stmt))

println("=== NULL ===")
stmt = null
println(checkString(stmt))
println(lengthString(stmt))
print(lengthStringDef(stmt))
```

?. -> safe calls

?: -> Elvis operator

!! -> null 허용하는 변수를 null이 아님을 보장

Null Safety

```kotlin
fun checkString(maybeString: String?): String = maybeString ?: "I THINK IT'S NULL"
fun lengthString(maybeString: String?): Int? = maybeString?.length
fun lengthStringDef(maybeString: String?): Int = maybeString!!.length
```

```kotlin
println("=== A ===")
var stmt: String? = "SELECT city AS 도시, COUNT(city)
        "  FROM user\n" +
        "    WHERE user.age >= 18\n" +
        "      GROUP BY city\n" +
        "        HAVING city >= 'b'\n" +
        "          ORDER BY city"
println(checkString(stmt))
println(lengthString(stmt))
println(lengthStringDef(stmt))

println("=== NULL ===")
stmt = null
println(checkString(stmt))
println(lengthString(stmt))
print(lengthStringDef(stmt))
```

```
=== A ===
SELECT city AS 도시, COUNT(city) AS 집계
  FROM user
    WHERE user.age >= 18
      GROUP BY city
        HAVING city >= 'b'
          ORDER BY city
144
144
=== NULL ===
I THINK IT'S NULL
null
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at MainKt.lengthStringDef(Main.kt:40)
    at MainKt.main(Main.kt:100)
```

Null Safety

```
Junsu Park
val birthDate: LocalDate
    get() = _birthDate!!.toLocalDate()

Junsu Park
private fun String.toLocalDate(): LocalDate =
    LocalDate.parse( text: this, DateTimeFormatter.ofPattern( pattern: "yyyy-MM-dd"))
```

Companion Object

클래스 내부에 정의되는 특별한 객체.
해당 클래스의 인스턴스를 생성하지 않고도 클래스의 멤버에 접근, 클래스 수준의 기능을 제공하는 데 사용

유틸리티 함수나 팩토리 메서드를 제공하거나 클래스 내의 상수값을 정의하는 데 사용

```
class MyClass {

    companion object {
        fun myFunction() {
            // ...
        }
    }
}
```

Extension functions

(클래스에 실제로 존재하는 메소드는 아님.) 기존에 정의된 클래스에 함수를 추가하는 기능

```kotlin
fun String.swapChar(i: Int, j: Int, filter: (Char) -> Boolean): String {
    val sb = StringBuilder(this)
    if (filter(sb[i]) && filter(sb[j])) {
        sb[i] = this[j]
        sb[j] = this[i]
    }
    return sb.toString()
}

fun main() {
    val str = "ABC"
    println(str.swapChar(0, 2))
}
```

02

Spring + kotlin

- java에서 kotlin으로..
- 생각할 점?

Kotlin의 수요....?

Spring Boot(Kotlin/Java)를 이용한 웹 애플리케이션 개발 경험이 있으신 분
RESTful API에 대한 이해와 지식, 경험이 있으신 분

Java 또는 Kotlin을 이용한 웹 애플리케이션 개발 경험이 4년 이상 있는 분
Web Framework(Spring Boot, Ktor)를 이용해서 웹 애플리케이션 개발/운영을 해본 분

Kotlin/Java + Spring/Spring Boot를 이용한 API 개발 경험이 있으신 분

[백엔드 개발자들의 기술 스택]
Spring Boot / Kotlin / Elastic Search / Spanner / Python

최소 3년 이상의 Spring framework 기반 JAVA/Kotlin Backend Engineer로서 근무 경험

자격 요건

• Java 또는 Kotlin 기반의 백엔드 개발 경험이

• C# 과 .NET(.NET Framework, .NET6, ASP.NET Core 6 Web API) 또는 Java/Kotlin과
Spring Boot를 사용하여 restful API 또는 daemon 개발 및 운영에 능숙

• Spring(Kotlin) 경험자 혹은 웹 서비스에 대한 이해가 있는 분

서로 비슷한 우리

Java

```java
package com.practice.reverselevel.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

1 usage  Junsu Park
@SpringBootApplication
public class Application {
    Junsu Park
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Kotlin

```kotlin
package com.example.authkt

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

Junsu Park
@SpringBootApplication
class AuthKtApplication

Junsu Park
fun main(args: Array<String>) {
    runApplication<AuthKtApplication>(*args)
}
```

그렇지만 다른 우리

class들은 기본적으로 final로 선언되며, open이라는 키워드를 붙여줘야만 상속 가능한 클래스가 된다

JPA 사용을 위해서는 아무런 argument를 받지 않는 기본 생성자가 필요하지만...

코틀린은 주 생성자를 만들 때 프로퍼티를 함께 만들어주는 방식을 사용하여 no-arg constructor가 존재 하지 않음

자바와는 쵸큼 다르군...

그렇지만 다른 우리

```
plugins {
    id 'org.springframework.boot' version '2.6.8'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
    id 'org.jetbrains.kotlin.jvm' version '1.6.21'
    id 'org.jetbrains.kotlin.plugin.jpa' version '1.6.21'
    id 'org.jetbrains.kotlin.plugin.spring' version '1.6.21'
}
```

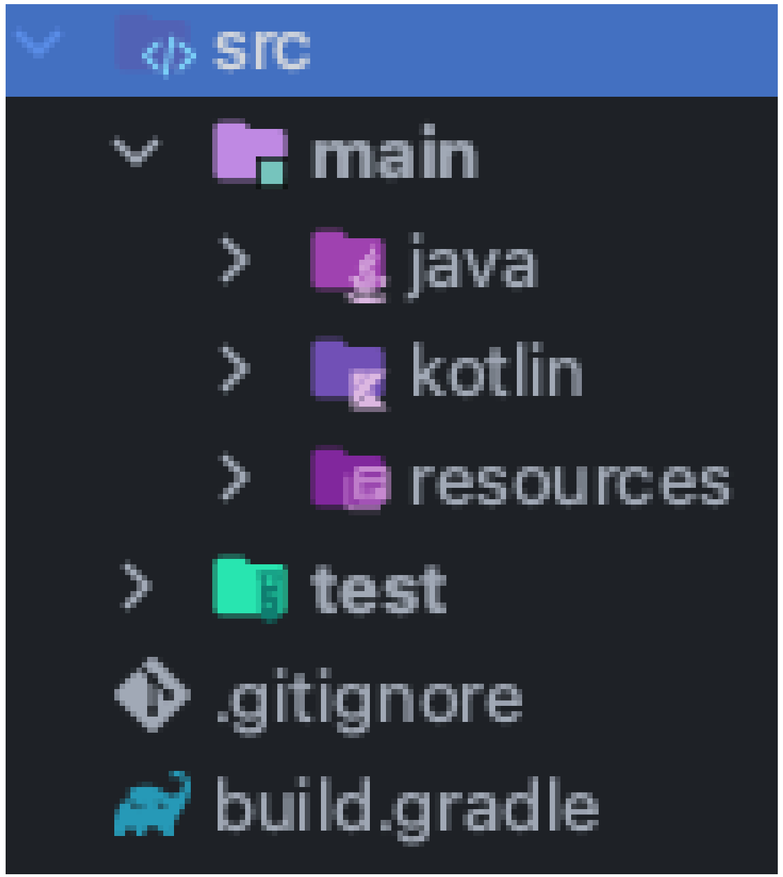
build.gradle의 초기 세팅 변경~

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'

    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'
    implementation 'org.jetbrains.kotlin:kotlin-reflect:1.6.21'
    implementation 'com.fasterxml.jackson.module:jackson-module-kotlin:2.13.3'

    runtimeOnly 'com.h2database:h2'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

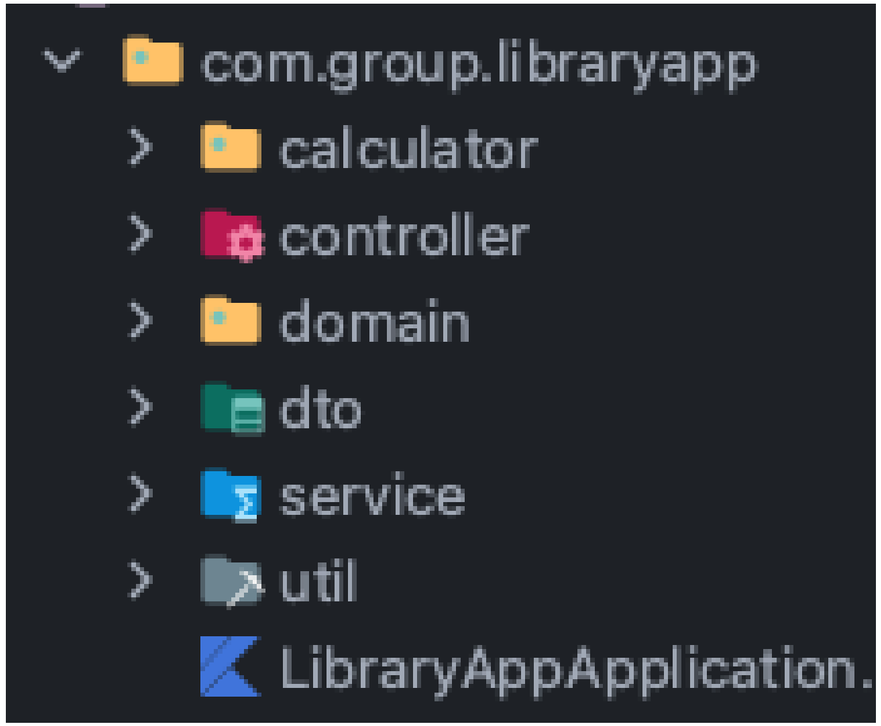
Jackson은 JSON 직렬화 및 역직렬화를 위한 강력한 라이브러리
Reflection은 클래스나 메서드 등을 런타임으로 제어하기 위한, 프로그램의 구조와 속성을 검사하고 조작할 수 있는 기술

```
compileKotlin {
    kotlinOptions {
        jvmTarget = "11"
    }
}

compileTestKotlin {
    kotlinOptions {
        jvmTarget = "11"
    }
}
```

파일 구조

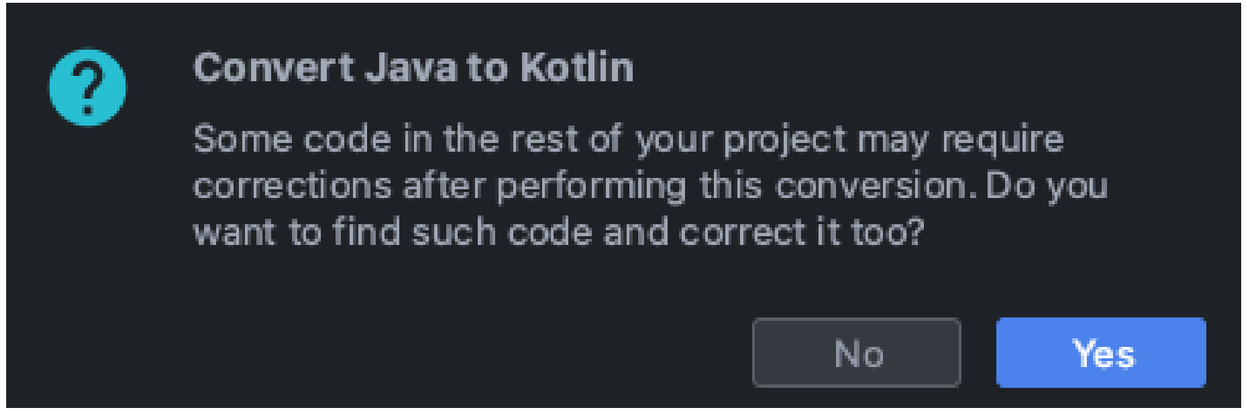


패키지 구조를 설정



DTOs

와 이런 엄청난 기능이??



DTOs

UserCreateRequest.java

```java
package com.group.libraryapp.dto.user.request;

public class UserCreateRequest {
  private String name;
  private Integer age;

  public UserCreateRequest(String name, Integer age) {
    this.name = name;
    this.age = age;
  }

  public String getName() {
    return name;
  }

  public Integer getAge() {
    return age;
  }
}
```

UserCreateRequest.kt

```kotlin
package com.group.libraryapp.dto.user.request

class UserCreateRequest(
    val name: String,
    val age: Int
)
```

EOF.

2023. 08. 31 Kotlin Spring
KIT SSL Junsu

DTOs

UserCreateRequest.kt

```kotlin
package com.group.libraryapp.dto.user.request

data class UserCreateRequest(
    val name: String,
    val age: Int?
)
```

data class를 사용하는 이유?

DTO의 기능과 의미를 한번 생각해보면? 언제든지 사용성이 있음 (테스트 등.)

그리고 내가 의도한건...

나이를 안 넣어도 상관없는데.. null이 될 수 있어야 함

인스턴스를 생성하는 방법 - Kotlin

1. 기본 생성자를 이용한 방법

```kotlin
class Person(val name: String, val age: Int)

val person = Person("Alice", 30)
```

2. 부생성자를 이용한 방법

```kotlin
class Person(val name: String, val age: Int){
      constructor(name: String) : this(name, 0)
}

val person = Person("Alice", 30)
```

3. companion object의 정적 메서드 방식

Kotlin은 static이 존재 하지 않음

```kotlin
class Person private constructor(val name: String, val age: Int) {

    companion object {
        fun of(name: String, age: Int): Person {
            return Person(name, age)
        }
    }
}

val person = Person.of("Alice", 30)
```

DTOs

객체 생성에 복잡한 로직이 필요하거나 객체의 생성 방법을 확장하고자 할 때
companion object의 정적 메서드 방식을 고려

```kotlin
data class UserResponse(
    val id: Long,
    val name: String,
    val age: Int?,
) {

    // 정적 팩토리 메서드 방식
    companion object {
        fun of(user: User): UserResponse{
            return UserResponse(
                id = user.id!!,
                name = user.name,
                age = user.age
            )
        }
    }
}
```

```kotlin
class UserResponse(
  val id: Long,
  val name: String,
  val age: Int?
) {

  constructor(user: User): this(
    id = user.id!!,
    name = user.name,
    age = user.age
  )

}
```

부생성자를 이용한 방법

DTOs

외부에서 생성자를 통한 호출이 불가능하고 새롭게 정의한 of라는 팩토리 메서드를 통해서만 인스턴화할 수 있음

UserService.java

```java
@Transactional(readOnly = true)
public List<UserResponse> getUsers() {
  return userRepository.findAll().stream()
      .map(UserResponse::new)
      .collect(Collectors.toList());
}
```

UserService.kt

```kotlin
@Transactional(readOnly = true)
fun getUsers(): List<UserResponse> {
    return userRepository.findAll().map { user -> UserResponse.of(user) }
}
```

Controllers (java)

```java
@RestController
public class BookController {

  private final BookService bookService;

  public BookController(BookService bookService) {
    this.bookService = bookService;
  }

  @PostMapping("/book")
  public void saveBook(@RequestBody BookRequest request) {
    bookService.saveBook(request);
  }

  @PostMapping("/book/loan")
  public void loanBook(@RequestBody BookLoanRequest request) {
    bookService.loanBook(request);
  }

  @PutMapping("/book/return")
  public void returnBook(@RequestBody BookReturnRequest request) {
    bookService.returnBook(request);
  }
}
```

```java
@RestController
public class UserController {

  private final UserService userService;

  public UserController(UserService userService) {
    this.userService = userService;
  }

  @PostMapping("/user")
  public void saveUser(@RequestBody UserCreateRequest request) {
    userService.saveUser(request);
  }

  @GetMapping("/user")
  public List<UserResponse> getUsers() {
    return userService.getUsers();
  }

  @PutMapping("/user")
  public void updateUserName(@RequestBody UserUpdateRequest request) {
    userService.updateUserName(request);
  }

  @DeleteMapping("/user")
  public void deleteUser(@RequestParam String name) {
    userService.deleteUser(name);
  }

}
```

Controllers (kotlin)

```kotlin
@RestController
class BookController(
    private val bookService: BookService,
) {

    Junsu Park
    @PostMapping("/book")
    fun saveBook(@RequestBody request: BookRequest) {
        bookService.saveBook(request)
    }

    Junsu Park
    @PostMapping("/book/loan")
    fun loadBook(@RequestBody request: BookLoanRequest) {
        bookService.loanBook(request)
    }

    Junsu Park
    @PutMapping("/book/return")
    fun returnBook(@RequestBody request: BookReturnRequest) {
        bookService.returnBook(request)
    }
}
```

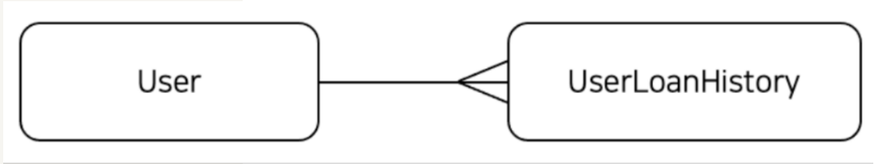
```kotlin
@RestController
class UserController(
    private val userService: UserService,
) {

    Junsu Park
    @PostMapping("/user")
    fun saveUser(@RequestBody request: UserCreateRequest) {
        userService.saveUser(request)
    }

    Junsu Park
    @GetMapping("/user")
    fun getUsers(): List<UserResponse> = userService.getUsers()


    Junsu Park
    @PutMapping("/user")
    fun updateUserName(@RequestBody request: UserUpdateRequest) {
        userService.updateUserName(request)
    }

    Junsu Park
    @DeleteMapping("/user")
    fun deleteUser(@RequestParam name: String) {
        userService.deleteUser(name)
    }
}
```

Entities

```java
@Entity
public class User {

  @Id
  @GeneratedValue(strategy = IDENTITY)
  private Long id;

  @Column(nullable = false)
  private String name;

  private Integer age;

  @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
  private final List<UserLoanHistory> userLoanHistories = new ArrayList<>();

  public User() {

  }

  public User(String name, Integer age) {
    if (name.isBlank()) {
      throw new IllegalArgumentException("이름은 비어 있을 수 없습니다");
    }
    this.name = name;
    this.age = age;
  }

  public void updateName(String name) { …
  }

  public void loanBook(Book book) {
    this.userLoanHistories.add(new UserLoanHistory(this, book.getName(), false));
  }

  public void returnBook(String bookName) {
    UserLoanHistory targetHistory = this.userLoanHistories.stream()
        .filter(history -> history.getBookName().equals(bookName))
        .findFirst()
        .orElseThrow();
    targetHistory.doReturn();
  }

  public String getName() { …
  }

  public Integer getAge() { …
  }
```

물론 Lombok을 사용하면 ....

Entities




```kotlin
@Entity
class User(

    var name: String,

    val age: Int?,

    @OneToMany(mappedBy = "user", cascade = [CascadeType.ALL], orphanRemoval = true)
    val userLoanHistories: MutableList<UserLoanHistory> = mutableListOf(),

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,
) {
    init {
        if (name.isBlank()) {
            throw IllegalArgumentException("이름은 비어 있을 수 없습니다")
        }
    }

    fun updateName(name: String) {
        this.name = name
    }

    fun loanBook(book: Book) {
        this.userLoanHistories.add(UserLoanHistory(this, bookName = book.name, false))
    }

    fun returnBook(bookName: String) {
        this.userLoanHistories
                .first { history -> history.bookName == bookName }
                .doReturn()
    }
}
```

User.kt

Setter

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,
) {
    init {
        if (name.isBlank()) {
            throw IllegalArgumentException("이름은 비어 있을 수 없습니다")
        }
    }

    fun updateName(name: String) {
        this.name = name
    }

    fun loanBook(book: Book) {
        this.userLoanHistories.add(UserLoanHistory(this, bookName = book.name, false))
    }

    fun returnBook(bookName: String) {
        this.userLoanHistories
                .first { history -> history.bookName == bookName }
                .doReturn()
    }
}
```

어디서 변경이 이루어졌는지 추적하기 쉽게 하기 위해서 setter 사용을 지양하는 방식으로 제작

setter를 바로 호출하는 것 보다는 좋을 것

그러나 Public으로 열려 있기 때문에

setter 호출도 외부에서 가능함..

Setter

UserResponse.java

```java
package com.group.libraryapp.dto.user.response;


import com.group.libraryapp.domain.user.User;

public class UserResponse {
  private final long id;
  private final String name;
  private final Integer age;

  public UserResponse(User user) {
    this.id = user.getId();
    this.name = user.getName();
    this.age = user.getAge();
  }

  public long getId() {
    return id;
  }

  public String getName() {
    return name;
  }

  public Integer getAge() {
    return age;
  }
}
```

Setter

```
class User(
  private var _name: String

){
  val name: String
    get() = this._name
}
```

backing property

private으로 get, set 둘 다 막고 getter 커스텀
(_name 프로퍼티, 읽기 전용 추가 프로퍼티 name)

```
class User(
  name: String
){

  var name = name
    private set

}
```

setter 커스텀

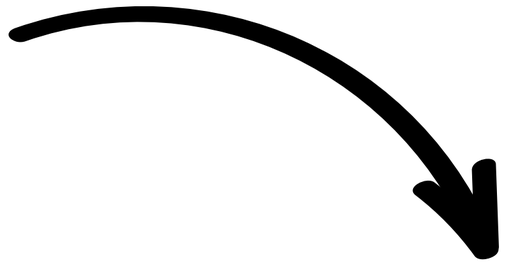
만약 Property들이 많아지게 된다면 감당 불가..... 현업에서는 팀간 컨벤션에 잘 맞춰하겠지만...

JPA Repository

Kotlin은 Optional이...

```
import org.springframework.data.jpa.repository.JpaRepository
import java.util.Optional

interface BookRepository : JpaRepository<Book, Long> {
    fun findByName(bookName: String): Optional<Book>
}
```

물론 사용은 가능합니다만.. java와 혼용하는 것? 별로야.



```
import org.springframework.data.jpa.repository.JpaRepository

interface BookRepository : JpaRepository<Book, Long> {
    fun findByName(bookName: String): Book?
}
```

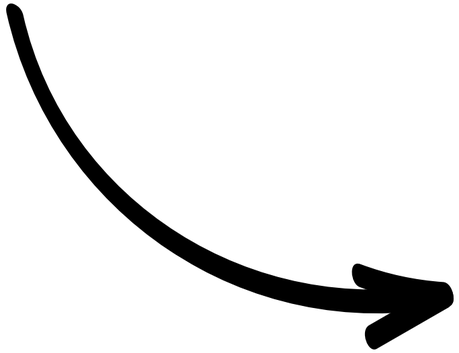
오.. 좋은데?

JPA Repository

```kotlin
@Transactional
fun deleteUser(name: String) {
    val user = userRepository.findByName(name)
                             .orElseThrow(::IllegalArgumentException)
    userRepository.delete(user)
}
```

ExceptionUtils.kt

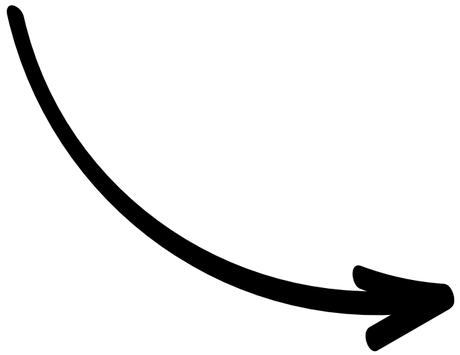
```kotlin
fun fail(): Nothing {
    throw IllegalArgumentException()
}
```



```kotlin
@Transactional
fun deleteUser(name: String) {
    val user = userRepository.findByName(name) ?: fail()
    userRepository.delete(user)
}
```

JPA Repository

```kotlin
@Transactional
fun updateUserName(request: UserUpdateRequest) {
    val user = userRepository.findById(request.id)
                             .orElseThrow(::IllegalArgumentException)
    user.updateName(request.name)
}
```



음... 바꾸고 싶은데..

ExceptionUtils.kt

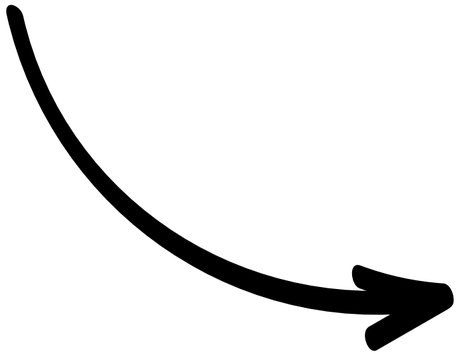
```kotlin
fun fail(): Nothing {
    throw IllegalArgumentException()
}
```

JPA Repository

```kotlin
@Transactional
fun updateUserName(request: UserUpdateRequest) {
    val user = userRepository.findById(request.id)
                             .orElseThrow(::IllegalArgumentException)
    user.updateName(request.name)
}
```

ExceptionUtils.kt

```kotlin
fun fail(): Nothing {
    throw IllegalArgumentException()
}
```



음... 바꾸고 싶은데..

```java
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
```

Optional 반환...?

```
1 implementation
Optional<T> findById(ID id);
```

JPA Repository

그래서 확장함수를 이용한다~ CrudRepositoryExtensions.kt 라는 것이 만들어져 있음!!

```
/*
 * Copyright 2018-2022 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.springframework.data.repository

Retrieves an entity by its id.
Params: id - the entity id.
Returns: the entity with the given id or null if none found
Authors: Sebastien Deleuze
Since: 2.1.4
fun <T, ID> CrudRepository<T, ID>.findByIdOrNull(id: ID): T? = findById(id).orElse( other: null)
```

ExceptionUtils.kt

```
fun fail(): Nothing {
    throw IllegalArgumentException()
}
```

그런데 우리는 null을 던지는 게 아니라
Exception까지 던질거야~

JPA Repository

요렇게 확장!

ExceptionUtils.kt

```kotlin
fun <T, ID> CrudRepository<T,ID>.findByIdOrThrow(id: ID): T {
    return this.findByIdOrNull(id) ?: fail()
}
```

데이터베이스에서 엔티티를 찾아오는 작업을 보다 안전하게 수행하도록 도와주는 확장 함수

cf. Generic

```
List list = new ArrayList();
list.add("Hello");
String value = (String) list.get(0);
```

특정 타입으로 제한하여 타입 안정성 제공

```
List<String> genericList = new ArrayList<>();
genericList.add("Hello");
String genericValue = genericList.get(0);
```

타입 체크, 형변환 생략 가능

cf. Generic

데이터의 타입(data type)을 일반화한다(generalize)는 것을 의미

클래스나 메서드에서 사용할 내부 데이터 타입을 외부에서 지정하는 방법

컴파일 시간에 자료형을 검사해 적당한 자료형을 선택할 수 있게 해주는 것

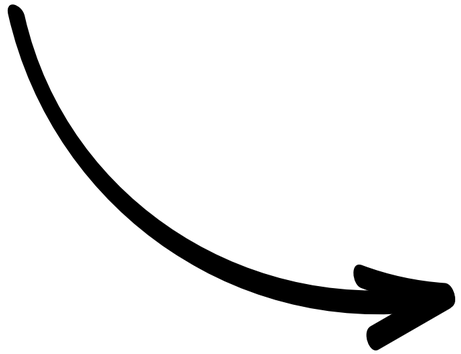
T는 엔티티 타입을, ID는 엔티티의 고유 식별자 타입

Generic Function - T, ID 타입을 명시 ==> <T, ID>

```
fun <T, ID> CrudRepository<T,ID>.findByIdOrThrow(id: ID): T {
    return this.findByIdOrNull(id) ?: fail()
}
```

JPA Repository

```kotlin
@Transactional
fun updateUserName(request: UserUpdateRequest) {
    val user = userRepository.findById(request.id)
                             .orElseThrow(::IllegalArgumentException)
    user.updateName(request.name)
}
```



```kotlin
@Transactional
fun updateUserName(request: UserUpdateRequest) {
    val user = userRepository.findByIdOrThrow(request.id)
    user.updateName(request.name)
}
```

오늘은 여기까지...

Fetch Join

N + 1 Query Problem

Coroutine

QueryDSL

출처

[1]https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/

[2] https://www.inflearn.com/pages/infcon-tech-kotlin

[3] https://colabear754.tistory.com/166

[4] 인프런 강의: 실전! 코틀린과 스프링 부트로 도서관리 애플리케이션 개발하기 (Java 프로젝트 리팩토링)

[5] https://multifrontgarden.tistory.com/272

[6] https://www.youtube.com/watch?v=m9aw5a50aDw

EOF.

2023. 08. 31 Kotlin Spring
KIT SSL Junsu